PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-215703
(43)Date of publication of application : 02.08.2002

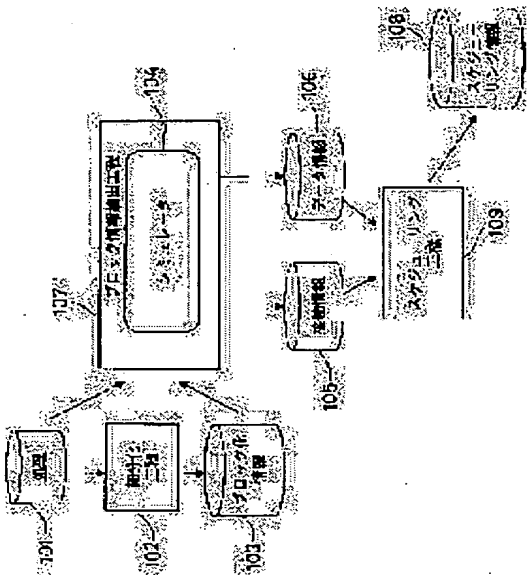
(51)Int.Cl. G06F 17/50
G06F 9/44
H01L 21/82

(21)Application number : 2001-008043
(22)Date of filing : 16.01.2001

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD
(72)Inventor : YONEDA KEI

(54) HARDWARE/SOFTWARE COOPERATIVE DESIGN METHOD

(57)Abstract:
PROBLEM TO BE SOLVED: To achieve requested performance in the minimum hardware constitution by restricting resources used in a hardware/software cooperative design method from an algorithm descriptive language.
SOLUTION: This method includes a dividing process 102 to divide a process 101 to blocks based on for-sentences and if-sentences to extract block generating information 103, a block information extracting process 107 to conduct simulation by a process simulator 104 based on block generating information and an execution time data base to determine connection information 105 to indicate data dependence relation between the blocks and data information 106 indicating data quantity renewed in processes in each block, and a scheduling process 109 to conduct scheduling for the blocks to the processes in the order of less holding data quantity based on the connection information and the data information for determining scheduling information 108.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－215703
（P2002－215703A）
(43)公開日　平成14年８月２日(2002.8.2)

(51)Int.Cl.'	識別記号	ＦＩ		テーマコード'（参考）
Ｇ０６Ｆ　17/50	６５４	Ｇ０６Ｆ　17/50	６５４Ｍ	５Ｂ０４６
9/44		9/06	６２０Ａ	５Ｂ０７６
Ｈ０１Ｌ　21/82		Ｈ０１Ｌ　21/82	Ｃ	５Ｆ０６４

審査請求　未請求　請求項の数４　ＯＬ　（全 12 頁）

(21)出願番号　特願2001－8043(P2001－8043)

(22)出願日　平成13年１月16日(2001.1.16)

(71)出願人　000005821
松下電器産業株式会社
大阪府門真市大字門真1006番地

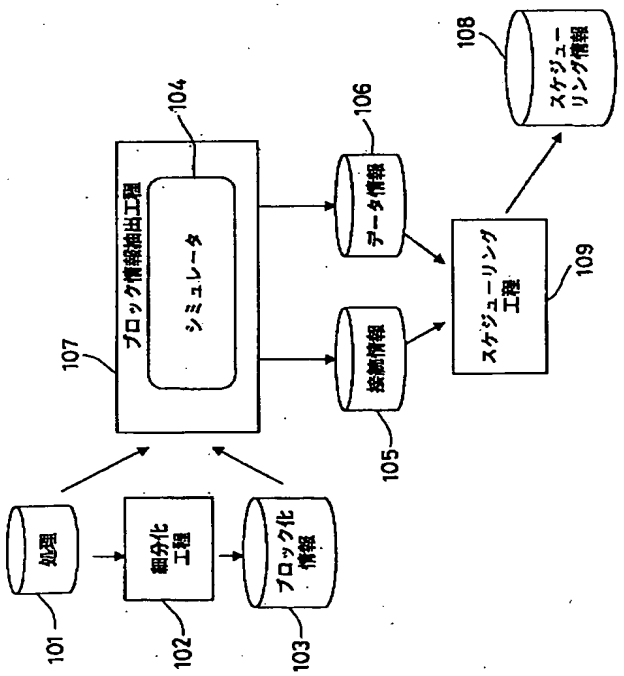
(72)発明者　米田　圭
大阪府門真市大字門真1006番地　松下電器産業株式会社内

(74)代理人　100095555
弁理士　池内　寛幸　（外５名）

Ｆターム(参考)　5B046 AA08 BA02 KA05
5B076 DD03 EC04
5F064 BB09 DD04 HH06 HH08 HH09
HH10 HH12

(54)【発明の名称】　ハードウェア／ソフトウェア協調設計方法

(57)【要約】　（修正有）
【課題】　アルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法において、使用する資源のリソースを抑え、最小限のハードウェア構成で要求性能を達成する。
【解決手段】　処理１０１をｆｏｒ文やｉｆ文でブロックごとに細分化しブロック化情報１０３を抽出する細分化工程１０２と、ブロック化情報及び実行時間データベースに基づいて処理のシミュレータ１０４によるシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報１０５および各ブロック内の処理において更新されたデータ量を示すデータ情報１０６を求めるブロック情報抽出工程１０７と、接続情報およびデータ情報に基づいて、保持するデータ量の少ない処理から順にブロックのスケジューリングを行ってスケジューリング情報１０８を算出するスケジューリング工程１０９とを含む。



1

【特許請求の範囲】

【請求項１】　アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報に基づいて前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報および各ブロック内の処理において更新されたデータ量を示すデータ情報を求めるブロック情報抽出工程と、

前記接続情報および前記データ情報に基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含むことを特徴とするハードウェア／ソフトウェア協調設計方法。

【請求項２】　前記スケジューリング工程において、

前記接続情報に基づいて、少なくとも２つの処理が並列に実行可能な分岐処理の条件式を検出し、

前記条件式の評価結果に従い実行される全ての処理に対して、前記データ情報を用いて、処理の実行中に保持する必要があるデータ量を算出し、

前記データ量が少ない処理から優先的に実行することを特徴とする請求項１記載のハードウェア／ソフトウェア協調設計方法。

【請求項３】　各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、

前記接続情報、前記データ情報、前記処理実行時間、および前記パイプライン処理のパイプラインピッチに基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含み、

前記パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とするハードウェア／ソフトウェア協調設計方法。

【請求項４】　各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

2

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、

前記処理実行時間および前記パイプライン処理のパイプラインピッチに基づいて、ハードウェアで実現するブロックを抽出し、ハードウェア化情報を算出するハードウェア化工程と、

前記接続情報、前記データ情報、前記処理実行時間、および前記ハードウェア化情報に基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含み、

ハードウェア化が必要なブロックについてはハードウェア化を実現しながら、前記パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とするハードウェア／ソフトウェア協調設計方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、処理のスケジューリングやハードウェア／ソフトウェアの割り当てに関するハードウェア／ソフトウェア協調設計方法に関する。

【０００２】

【従来の技術】Ｃ／Ｃ＋＋などのアルゴリズム記述言語で記述されたシステム全体の処理に対し、ＡＳＩＣ等にハードウェア化する部分、ＣＰＵで実行させるソフトウェア部分などとシステム全体の処理をハードウェア／ソフトウェアに割り当てる必要がある。いわゆるハードウェア／ソフトウェア協調設計である。ハードウェア／ソフトウェア協調設計では、システム全体の処理を適当なブロックでブロック化しスケジューリングしたものに対し、ハードウェア／ソフトウェアの割り当てを行う。その際に、従来ではハードウェア／ソフトウェア協調設計を行う明確な指針／方法がないため、既存の設計データの流用や設計者の経験や勘に頼るところが大きい。

【０００３】

【発明が解決しようとする課題】まず、従来のハードウェア／ソフトウェア協調設計において既存の設計データを流用した場合、初期の設計コストは減少するが、既存の設計データの仕様制限などを受けるため、システム全

体の性能や面積等を考慮した最適なハードウェア／ソフトウェア協調設計を行うことは困難である。そのため、設計段階が進むにつれ新規システムに対する要求性能を満足させることができずに、ハードウェア／ソフトウェア協調設計のやり直しが発生する場合があり、設計コストや設計期間が増大する可能性がある。特に、既存の設計データがハードウェアである場合、そのブロックについては性能、面積、消費電力などが固定値であり、仕様変更などに対して柔軟な対応がとれない。

【０００４】また、設計者の経験や勘では、従来までの方法の繰り返しになる場合が多く、システムごとの最適なハードウェア／ソフトウェア協調設計が行われず、性能、面積などの向上が思い通りに図れないことになる。

【０００５】つまり、ハードウェア／ソフトウェア協調設計の明確な指針／方法が無いため、設計プロセスの発展に反して既存データ／設計方法を流用することになり、最適なハードウェア／ソフトウェア協調設計が行われていないというのが実情である。

【０００６】本発明は、上記従来の問題点に鑑みてなされたものであり、その目的は、対象とする処理を主にソフトウェアで行なうシステム処理に対して、使用する資源を抑えることが可能なハードウェア／ソフトウェア協調設計方法を提供することにある。

【０００７】また、本発明の他の目的は、マルチプロセッサを用いたパイプライン処理によるシステム処理に対して、必要に応じてハードウェアに処理を割り当てながら、使用する資源を抑えかつ要求性能を満たすことが可能なハードウェア／ソフトウェア協調設計方法を提供することにある。

【０００８】

【課題を解決するための手段】前記の目的を達成するため、本発明に係る第１のハードウェア／ソフトウェア協調設計方法は、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報に基づいて前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報および各ブロック内の処理において更新されたデータ量を示すデータ情報を求めるブロック情報抽出工程と、接続情報およびデータ情報に基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含むことを特徴とする。

【０００９】この場合、スケジューリング工程において、接続情報に基づいて、少なくとも２つの処理が並列に実行可能な分岐処理の条件式を検出し、条件式の評価結果に従い実行される全ての処理に対して、データ情報を用いて、処理の実行中に保持する必要があるデータ量を算出し、データ量が少ない処理から優先的に実行することが好ましい。

【００１０】この第１のハードウェア／ソフトウェア協調設計方法によれば、細分化しブロック化した処理に対し、各ブロックで更新されたデータ量を評価しながらスケジューリングを行うことで、記憶装置のリソースを抑えることができる。

【００１１】前記の目的を達成するため、本発明に係る第２のハードウェア／ソフトウェア協調設計方法は、各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、接続情報、データ情報、処理実行時間、およびパイプライン処理のパイプラインピッチに基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含み、パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とする。

【００１２】この第２のハードウェア／ソフトウェア協調設計方法によれば、マルチプロセッサのパイプライン処理でシステムのスケジューリングを行う際に、細分化した各ブロックの処理に要する実行時間を算出し、パイプラインピッチに過不足なくブロックをスケジューリングすることで、各パイプラインステージにおけるプロセッサの遊びの時間を少なくすることができる。

【００１３】前記の目的を達成するため、本発明に係る第３のハードウェア／ソフトウェア協調設計方法は、各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、処理実行時間およびパイプライン処理のパイプラインピッチに基づいて、ハードウェアで実現するブロックを抽出し、ハードウェア化情報を算出するハードウェア化工程と、接続情報、データ情報、処理実行

時間、およびハードウェア化情報に基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含み、ハードウェア化が必要なブロックについてはハードウェア化を実現しながら、パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とする。

【００１４】この第３のハードウェア／ソフトウェア協調設計方法によれば、ハードウェア化が必要なブロックを抽出し、ハードウェア化を実現しながらスケジューリングを行うことで、対象となるシステムに対して、最小限のハードウェア構成で要求性能を満たすことができる。

【００１５】

【発明の実施の形態】以下、本発明の実施の形態について、図面を参照して説明する。

【００１６】（第１の実施の形態）図１は、本発明の第１の実施の形態によるアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成を模式的に示す図である。図１において、ハードウェア／ソフトウェア協調設計方法は、アルゴリズム記述言語で記述された処理１０１と、処理を細分化しブロック化する細分化工程１０２と、細分化工程１０２より得られる処理のブロック化情報１０３と、ブロック化情報１０３をもとにシミュレータ１０４による処理１０１のシミュレーションを行い、ブロック間の接続情報１０５とブロックで更新されるデータ量を示すデータ情報１０６を抽出するブロック情報抽出工程１０７と、接続情報１０５およびデータ情報１０６に基づいて、ブロックレベルでのスケジューリングを行い、スケジューリング情報１０８を算出するスケジューリング工程１０９から構成される。

【００１７】図２は、本実施形態においてスケジューリング対象となっているアルゴリズム記述言語（Ｃ言語）で記述された処理１０１の一例を示す図である。

【００１８】図３および図４は、図２の処理を細分化工程１０２より細分化した時のブロック化情報１０３を表す図である。

【００１９】図５は、ブロック情報抽出工程１０７から得られる図２の処理の接続情報１０５を表す図である。

【００２０】図６は、ブロック情報抽出工程１０７から得られる図２の処理のデータ情報１０６を表す図である。

【００２１】図７は、図５の接続情報と図６のデータ情報を統合して表した図である。

【００２２】図８および図９は、分岐処理の実行順序を変えた場合における各ブロックの処理結果を保持するのに最低限必要な記憶装置の容量を表わした図である。

【００２３】図１０は、スケジューリング工程１０９から算出されるスケジューリング情報１０８を表す図である。

【００２４】次に、本実施の形態によるハードウェア／ソフトウェア協調設計方法について、図１から図１０を用いて具体的に説明する。

【００２５】図２で示す処理は、ｍａｉｎ関数の１４行目において、サブモジュールであるｆｕｎｃｔｉｏｎ関数を呼んでいる算術演算の処理を表す。この処理に対してスケジューリングを行うには、まず処理のブロック化が必要である。細分化工程１０２では、構文解析を行ってｆｏｒ文やｉｆ文で示される分岐処理の記述とその他の代入文で細分化を行う。つまり図２の処理において、ｆｏｒ文、ｉｆ文、ｅｌｓｅ文などの分岐処理の条件式毎に細分化を行う。

【００２６】図２のｍａｉｎ関数およびｆｕｎｃｔｉｏｎ関数の細分化結果であるブロック化情報１０３をそれぞれ図３および図４に示す。図３では、７行目のｆｏｒ文、８行目のｉｆ文、１１行目のｅｌｓｅ文で、細分化を行われ、さらに１１行目のｅｌｓｅ文の処理が終了する１３行目、７行目のｆｏｒ文の処理が終了する１５行目で細分化が行われている。図４についても同様に、ｆｕｎｃｔｉｏｎ関数について細分化を行っている。なお、図３および図４では、以降の説明のために細分化したブロック毎にＡ１、Ａ２、…およびＢ１、Ｂ２、…とブロック番号を付加している。

【００２７】次に、図２で示す処理１０１と、図３および図４で示すブロック化情報１０３とを、ブロック情報抽出工程１０７に入力し、接続情報１０５およびデータ情報１０６を抽出する。以下にその工程を説明する。

【００２８】図５は、図４のブロック化情報１０３を用いてブロック情報抽出工程１０７より得られた各ブロックごとのデータ依存関係である接続情報１０５を示す。図５では、ｆｕｎｃｔｉｏｎ関数の処理を逐次的に処理した場合の実行順に矢印（斜線）を記載している。この図において、ブロックＢ２の処理結果である“ｚ１［０］～ｚ１［１５］”のデータはブロックＢ４で使用されるため、処理の接続情報としてブロックＢ２からブロックＢ４へ矢印（実線）が記載されている。以下同様に、ブロックＢ３からブロックＢ５へはデータ“ｃ”が、ブロックＢ４からブロックＢ６へはデータ“ｄ”が、ブロックＢ５からブロックＢ６へはデータ“ｚ２［０］～ｚ２［１５］”が処理結果として受け渡されている。

【００２９】次に、図６は、図４のブロック化情報１０３を用いてブロック情報抽出工程１０７より得られる各ブロックで更新されたデータ量であるデータ情報１０６を示す。例えば、図６において、ブロックＢ１で更新されるデータ量は“ｃ”、“ｄ”、“ｅ”であり、データ量は３となる。以下同様に更新されるデータ量は、ブロックＢ２では“ｚ１［０］”～“ｚ１［１５］”で１６、ブロックＢ３では“ｃ”で１、ブロックＢ４では“ｄ”で１、ブロックＢ５では“ｚ２［０］”～“ｚ２

［１５］”で１６、ブロックＢ６では“ｅ”で１となる。ここでデータ量の換算方法は整数１個を１としているが、処理によって換算方法は自由に設定できることはいうまでもない。

【００３０】図５の接続情報１０５から、ブロックＢ２で確定したデータ“ｚ１［０］”～“ｚ１［１５］”の値がブロックＢ４で読み出され、ブロックＢ３で確定したデータ“ｃ”がブロックＢ５で読み出されていることがわかる。さらに、ブロックＢ２とブロックＢ３の間にはデータの確定／読み出しの関係がないことがわかるため、ブロックＢ２、Ｂ４の処理とブロックＢ３、Ｂ５の処理が互いに並列処理可能であることがわかる。つまり、ブロックＢ１の後にブロックＢ２とブロックＢ３はどちらの処理を先に行っても問題ない。さらに、ブロックＢ６は、図５よりブロックＢ４とブロックＢ５の処理結果を読み出しているため、ブロックＢ４、ブロックＢ５の処理が終了しなければブロックＢ６は処理が開始できない。

【００３１】以上のブロック間のデータ依存関係に、図６のデータ情報１０６を加えたものを図７に示す。図７において、ブロックＢ２、Ｂ４の処理とブロックＢ３、Ｂ５の処理が並列処理可能であり、ブロックＢ６はこの２つの並列処理の結果を読み出して処理していることがわかる。さらに、図７には、ブロック間のデータ情報も付記して示しており、例えばブロックＢ２で確定されたデータ“ｚ１［０］”～“ｚ１［１５］”のデータ量１６がブロックＢ４で読み出されていることがわかる。

【００３２】以下では、図２の関数ｆｕｎｃｔｉｏｎを例に、スケジューリング工程１０９について説明する。

【００３３】関数ｆｕｎｃｔｉｏｎの処理を１つのプロセッサで処理する場合、図７から分かるように、ブロックＢ１の次にブロックＢ２、Ｂ４あるいはブロックＢ３、Ｂ５のどちらのブロックを優先的に処理するかが問題となる。

【００３４】図８は、処理の順番をブロックＢ１、Ｂ２、Ｂ４、Ｂ３、Ｂ５、Ｂ６（以下順番Ｃとする）とした場合における、各ブロックの処理結果を保持するために最低限必要な記憶装置の容量を表したものである。同様に、図９は、処理の順番をブロックＢ１、Ｂ３、Ｂ５、Ｂ２、Ｂ４、Ｂ６（以下順番Ｄとする）とした場合における、各ブロックの処理結果を保持するために最低限必要な記憶装置の容量を表している。図８、図９ともに、当該ブロックが次のブロックへ処理する際に最低限保持しなくてはならないデータのための記憶装置の容量をそれぞれ明記している。

【００３５】まず、図８について説明する。図８のブロックＢ１－Ｂ２間においてデータ“ｃ”、“ｄ”、“ｅ”が、またブロックＢ２－Ｂ４間においてデータ“ｚ１［０］”～“ｚ１［１５］”が新規に更新され、これらのデータを以降の処理で必要なデータとして保持する必要がある。しかし、Ｂ４－Ｂ３間では、データ“ｄ”が新規に更新されるが、ブロックＢ１－Ｂ２間においてデータ“ｄ”を保持する記憶装置は既に用意されているため、データ“ｄ”を保持するための記憶装置は新たに必要でない。ブロックＢ３－Ｂ５間で更新されるデータ“ｃ”についても同様である。そして、ブロックＢ５－Ｂ６間では、データ“ｚ２［０］”～“ｚ２［１５］”が新たに更新され、それを保持する記憶装置が必要となる。

【００３６】次に、図９について説明する。図９では、ブロックＢ１－Ｂ３間において、データ“ｃ”、“ｄ”、“ｅ”が新規に更新され、これらのデータを以降の処理で必要なデータとして保持する必要がある。しかし、次のブロックＢ３－Ｂ５間では、データ“ｃ”が更新されるが、ブロックＢ１－Ｂ３間でデータ“ｃ”を保持する記憶装置は既に用意されているので、データ“ｃ”を保持するための記憶装置は新たに必要ではない。次のブロックＢ５－Ｂ２間では、データ“ｚ２［０］”～“ｚ２［１５］”を保持するための記憶装置が新たに必要で、さらにブロックＢ２－Ｂ４間では、データ“ｚ１［０］”～“ｚ１［１５］”を保持するための記憶装置が必要となる。しかし、後の処理であるブロックＢ４－Ｂ６間では、データ“ｚ１［０］”～“ｚ１［１５］”は必要ないためデータ“ｚ１［０］”～“ｚ１［１５］”を保持する記憶装置は必要なくなる。

【００３７】以上、図８および図９から、処理結果を一時的に保持する記憶装置の容量は、順番Ｃで処理を行ったほうが順番Ｄで処理を行うよりも小さいことがわかる。よって記憶装置のリソースの観点から、各ブロックの処理を順番Ｃで行った方が良いことがわかる。以上より、スケジューリング工程１１０から得られる関数ｆｕｎｃｔｉｏｎのスケジューリング情報１０９は、図１０に示すものとなり、本実施の形態によれば、記憶装置のリソースが少なくなるように、ブロックのスケジューリングを行なうことが可能になる。

【００３８】なお、以上のスケジューリング方法は、１つのプロセッサの場合に限らず複数のプロセッサで処理する場合においても同様であることは言うまでも無い。

【００３９】（第２の実施の形態）図１１は、本発明の第２の実施の形態によるアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成を模式的に示す図である。図１１には、複数の同一のプロセッサが各々命令メモリを備え、このプロセッサが各パイプラインステージの処理を実行するマルチプロセッサのパイプライン処理において、本実施の形態によるハードウェア／ソフトウェア協調設計方法を示している。

【００４０】図１１において、ハードウェア／ソフトウェア協調設計方法は、特定のプロセッサにおける各命令の実行時間をデータベースとしてまとめた命令実行時間のデータベース１１０１と、パイプライン処理の対象と

なる処理１１０２と、細分化工程１１０３より細分化したブロック化情報１１０４および命令実行時間のデータベース１１０１に基づいて、処理１１０２のシミュレータ１１０５によるシミュレーションを行ない、ブロック間の接続情報１１０６、ブロック内で更新されるデータ量を示すデータ情報１１０７、および各ブロックごとの処理に要する時間である処理実行時間１１０８を抽出するブロック情報抽出工程１１０９と、接続情報１１０６、データ情報１１０７、処理実行時間１１０８、および設計者の要求するパイプラインピッチ１１１０に基づいてブロックレベルでのスケジューリングを行い、スケジューリング情報１１１１を算出するスケジューリング工程１１１２とから構成される。

【００４１】図１２は、処理１１０２における接続情報１１０６と処理実行時間１１０８の一例を表した図である。

【００４２】図１３は、スケジューリング工程１１１２から得られるスケジューリング情報１１１１を示す図である。

【００４３】図１４は、本実施形態においてスケジューリング可能か否かの判断手順を示すフローチャートである。

【００４４】次に、本実施の形態によるハードウェア／ソフトウェア協調設計方法について、図１１から図１４を用いて具体的に説明する。

【００４５】図１１において、命令実行時間のデータベース１１０１は、特定のプロセッサの命令、例えば加算、積算などの単純な命令ごとに処理時間をデータベースとしてまとめたものである。これをブロック化情報１１０４とともにシミュレーションすることで、各ブロックにおける処理実行時間１１０８を命令実行時間のデータベース１１０１を参照して算出することができる。ここで、ブロック化情報１１０４、接続情報１１０６、データ情報１１０７は第１の実施の形態で説明した手順と同様な方法で抽出可能である。

【００４６】図１２は、ブロック情報抽出工程１１０９から得られる処理実行時間１１０８と接続情報１１０６を表す。図１２では、処理１１０２を細分化工程１１０３より細分化し、１つ以上のブロックで構成される一連の処理ごとにａ、ｂ、ｃ、…とし、さらにその一連の処理においてブロック毎にａ１、ａ２、ｂ１、ｂ２、…としている。さらに、各ブロック（ａ１、ａ２、ｂ１、ｂ２、…）の長さをそれぞれの処理実行時間１１０８の大きさに対応させ、並列に実行可能な処理については各ブロックを並列に表記している（例えば、ｅ１、ｅ２とｆ１、ｆ２は並列に処理可能である）。

【００４７】次に、図１２で示す処理実行時間１１０８と接続情報１１０６をもつ処理１１０２に対するスケジューリングの方法について説明する。

【００４８】まず、パイプラインピッチ１１１０をスケジューリング工程１１１２に入力する必要がある。ここで、パイプラインピッチ１１１０は、パイプライン処理の各ステージで行う処理実行時間を表しており、各ステージの処理実行時間を均一化し、ばらつきを無くすことにより、各ステージの処理に遊びの時間がなくなり、高速なパイプライン処理の設計が可能となる。そこで設計者は、要求する処理性能とパイプライン処理のステージ数からパイプラインピッチ１１１０を算出し、それをスケジューリング工程１１１２に入力する。なお、本実施の形態におけるマルチプロセッサのパイプライン処理では、パイプライン処理のステージ数は使用するプロセッサの個数と等価である。

【００４９】図１２に示すデータ依存関係および処理実行時間を有する処理から、ブロックａ１、ａ２の次に処理可能なブロックの候補として、ブロックｂ１～ｂ４、ｃ１～ｃ２、およびｄ１～ｄ４があることがわかる。ここで、ブロックａ１、ａ２の処理実行時間の和がパイプラインピッチ１１１０よりも小さい場合、つまり同一のパイプラインピッチ１１１０内にブロックａ１、ａ２の他にさらにブロックの追加が可能である場合、ブロックｂ１～ｂ４、ｃ１～ｃ２、およびｄ１～ｄ４の中からいずれかのブロックを選択し追加しなくてはならない。

【００５０】本実施の形態によるスケジューリング工程１１１２では、ブロックａ１、ａ２の次にブロックｂ１またはブロックｃ１あるいはブロックｄ１を挿入し、その時の処理実行時間の総和をそれぞれの場合において算出する。そして、処理実行時間の総和がパイプラインピッチ１１１０よりも小さい場合には、さらに処理を挿入する。例えば、ブロックａ１、ａ２、ｂ１の処理実行時間の総和がパイプラインピッチ１１１０よりも小さい場合には、ブロックｂ１に継続して処理されるブロックｂ２を挿入し、その時の処理実行時間の総和を算出する。そして算出した処理実行時間の総和がパイプラインピッチ１１１０よりも大きければ、パイプラインピッチ１１１０に挿入可能な処理はブロックａ１、ａ２、ｂ１であると確定し、逆に、パイプラインピッチ１１１０よりも小さければさらにブロックを追加する。

【００５１】以上の作業を繰り返すことにより、パイプラインステージにおける処理実行時間の総和がパイプラインピッチ１１１０よりも大きくならない程度まで可能な限り多くのブロックを挿入する。これを並列処理可能なブロックごとに行い算出した処理実行時間の総和を比較して、最もパイプラインピッチ１１１０に近い値をもつブロックの順番をスケジューリング情報１１１１として算出する。

【００５２】図１３のスケジューリング情報１１１１は、ブロックａ１、ａ２の次にブロックｄ１、ｄ２を挿入した時の処理実行時間がパイプラインピッチ１１１０に最も近い値になったため、ブロックａ１、ａ２、ｄ１、ｄ２をパイプライン処理の第１のステージに行う処

理として確定している。以降、第２のステージでは、第１のステージにおける処理の継続としてブロックｄ３、ｄ４の処理を実行し、その後の処理については、ブロックｃ１～ｃ２とブロックｂ１～ｂ４の中からパイプラインピッチ１１１０に最も過不足なく収まるものを選択し、スケジューリングを行っている。

【００５３】なお、上記のスケジューリング方法において、当該ブロックの次に実行する処理が一意に決定できない場合、例えばパイプラインステージの処理としてａ１、ａ２、ｄ１、ｄ２を行なう場合とａ１、ａ２、ｃ１を行なう場合で処理実行時間の総和が等しい場合には、第１の実施の形態と同様に、処理結果を保持する記憶装置のリソースが少ない処理を、次に処理するブロックとして選択する。

【００５４】なお、本実施の形態によるハードウェア／ソフトウェア協調設計方法では、パイプラインピッチ１１１０よりも処理実行時間が長いブロックがある場合には、本実施の形態のスケジューリングは実行できない。

【００５５】図１４は、本実施の形態によるスケジューリングが可能か否かの判断を行うフローチャートを示す。各ブロックの処理実行時間Ｔｓとパイプラインピッチ１１１０（Ｐｐ）を比較し、その比較の結果、パイプラインピッチ１１１０よりも処理時間を要するブロックが１つでもあった場合、本実施の形態のスケジューリングは実行できない。これを解決する方法について、第３の実施の形態として次に説明する。

【００５６】（第３の実施の形態）図１５は、本発明の第３の実施の形態によるアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成を模式的に示す図である。図１５には、複数の同一のプロセッサが各々命令メモリを備え、このプロセッサが各パイプラインステージの処理を実行するマルチプロセッサのパイプライン処理において、本実施の形態によるハードウェア／ソフトウェア協調設計方法を示している。　図１５において、ハードウェア／ソフトウェア協調設計方法は、特定のプロセッサにおける各命令の実行時間をデータベースとしてまとめた命令実行時間のデータベース１５０１と、パイプライン処理の対象となる処理１５０２と、細分化工程１５０３より細分化したブロック化情報１５０４および命令実行時間のデータベース１５０１に基づいて、処理１５０２のシミュレータ１５０５によるシミュレーションを行ない、ブロック間の接続情報１５０６、ブロック内で更新されるデータ量を示すデータ情報１５０７、および各ブロックごとの処理に要する時間である処理実行時間１５０８を抽出するブロック情報抽出工程１５０９と、処理実行時間１５０８および設計者の要求するパイプラインピッチ１５１０から、ハードウェア化が必要なブロックを抽出し、ハードウェア化情報１５１１を抽出するハードウェア化工程１５１２と、接続情報１５０６、データ情報１５０７、処理実行時間１５０８、ハードウェア化情報１５１１、およびパイプラインピッチ１５１０から、ブロックレベルでのスケジューリングを行い、スケジューリング情報１５１３を算出するスケジューリング工程１５１４とから構成される。

【００５７】図１６は、本実施形態のハードウェア化工程１５１２で行われる処理手順を示すフローチャートである。

【００５８】次に、本実施の形態によるハードウェア／ソフトウェア協調設計方法について、図１５および図１６を用いて具体的に説明する。なお、本実施の形態によるハードウェア／ソフトウェア協調設計方法は、第２の実施の形態に、パイプラインピッチ１５１０および処理実行時間１５０８からハードウェア化が必要なブロックを抽出し、ハードウェア化を行うハードウェア化工程１５１２が追加されたものである。

【００５９】図１６に示すように、パイプラインピッチ１５１０（Ｐｐ）と処理実行時間１５０８（Ｔｓ）を比較し、パイプラインピッチ１５１０よりも処理時間が長いブロックについては、パイプラインピッチ１５１０に収まるように高位合成などによりハードウェア化を行う。図１５のハードウェア化情報１５１１は、この時のハードウェア化されたブロックの情報を示す。このハードウェア化情報１５１１に基づき、スケジューリング工程１５１４では、ハードウェア化されたブロックを検出し、１つのパイプラインステージ内にハードウェア化されたブロックを割り当てる。それ以外のハードウェア化が必要ないブロックについては、第２の実施の形態と同様のハードウェア／ソフトウェア協調設計を行う。

【００６０】このように、本実施の形態によれば、最小限のハードウェア構成で要求する性能を満たすような処理のハードウェア／ソフトウェア協調設計が行える。ハードウェア構成を極力少なくすることにより、仕様変更などに対してもプロセッサで処理させるプログラムの変更などで柔軟に対応することができる。

【００６１】

【発明の効果】以上説明したように、本発明によれば、細分化しブロック化した処理に対し、各ブロックで更新されたデータ量を評価しながらスケジューリングを行うことで、記憶装置のリソースが少ないハードウェア／ソフトウェア協調設計が可能になる。

【００６２】また、マルチプロセッサのパイプライン処理でシステムのスケジューリングを行う際に、細分化した各ブロックの処理に要する実行時間を算出し、パイプラインピッチに過不足なくブロックをスケジューリングすることで、各パイプラインステージにおけるプロセッサの遊びの時間が少ないハードウェア／ソフトウェア協調設計が可能になる。

【００６３】さらに、ハードウェア化が必要なブロックを抽出し、ハードウェア化を実現しながらスケジューリングを行うことで、対象となるシステムに対して、最小

限のハードウェア構成で要求性能を満たすハードウェア／ソフトウェア協調設計が可能になる。

【図面の簡単な説明】

【図１】　本発明の第１の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

【図２】　図１のアルゴリズム記述言語で記述された処理１０１の一例を示す図

【図３】　図２のｍａｉｎ関数のブロック化情報１０３を模式的に示す図

【図４】　図２のｆｕｎｃｔｉｏｎ関数のブロック化情報１０３を模式的に示す図

【図５】　図１の接続情報１０５を模式的に示す図

【図６】　図１のデータ情報１０６を模式的に示す図

【図７】　図１の接続情報１０５およびデータ情報１０６を模式的に示す図

【図８】　図１の処理１０１が実行順序Ｃの場合に必要な記憶装置の容量を示す図

【図９】　図１の処理１０１が実行順序Ｄの場合に必要な記憶装置の容量を示す図

【図１０】　図１のスケジューリング情報１０８を模式的に示す図

【図１１】　本発明の第２の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

【図１２】　図１１の接続情報１１０６および処理実行

時間１１０８を模式的に示す図

【図１３】　図１１のスケジューリング情報１１１１を模式的に示す図

【図１４】　本発明の第２の実施の形態におけるスケジューリングが可能か否かの判断手順を示すフローチャート

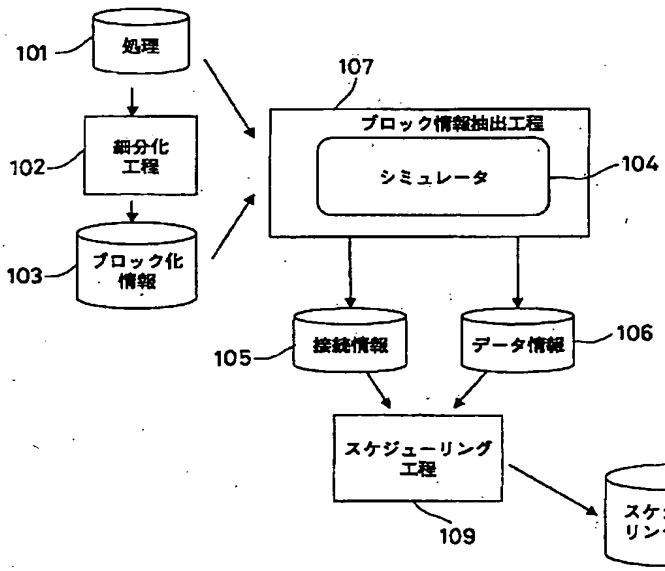
【図１５】　本発明の第３の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

【図１６】　図１５のハードウェア化工程１５１４における処理手順を示すフローチャート

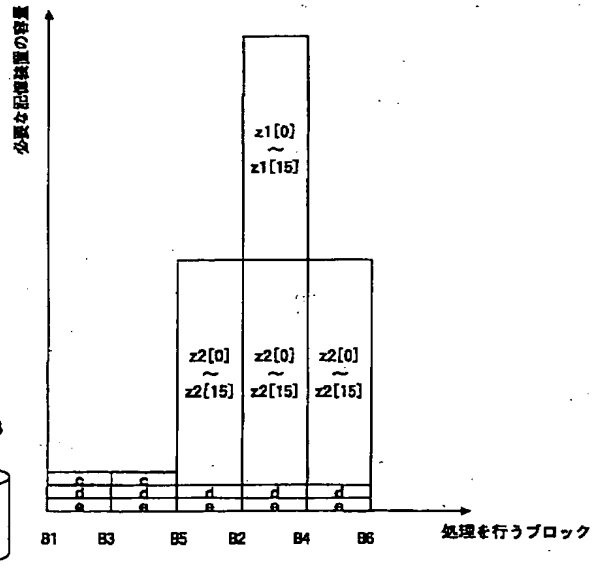
【符号の説明】

１０１、１１０２、１５０２　処理
１０２、１１０３、１５０３　細分化工程
１０３、１１０４、１５０４　ブロック化情報
１０４、１１０５、１５０５　シミュレータ
１０５、１１０６、１５０６　接続情報
１０６、１１０７、１５０７　データ情報
１０７、１１０９、１５０９　ブロック情報抽出工程
１０８、１１１１、１５１３　スケジューリング情報
１０９、１１１２、１５１４　スケジューリング工程
１１０１、１５０１　命令実行時間データベース
１１０８、１５０８　処理実行時間
１１１０、１５１０　パイプラインピッチ
１５１１　ハードウェア化情報
１５１２　ハードウェア化工程

【図１】



【図９】



【図２】

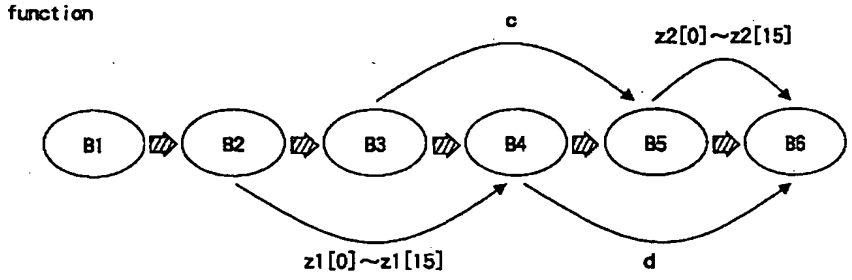
```
1: main()
2: {
3:          int i, a, b, x[16], y[16];
4:          int x={2,7,3,6,9,10,1,5,6,6,13,17,9,4,7,8}
5:          int y={1,4,7,2,5,8,3,6,9,15,16,1,7,7,0,0}
6:          a=0;
7:          for (i=0;i<16;i++){
8:                    if(i<8){
9:                              b=a+2;
10:                   }
11:                   else{
12:                             b=a+1;
13:                   }
14:                   function(a,b,x,y);
15:         }
16: }
```

```
1: function(a,b,x,y)
2: {
3:          int j, k, l, m, n, c, d, e, z1[16], z2[16];
4:          c=0;
5:          d=0;
6:          e=0;
7:          for(j=0;j<16;j++){
8:                    z1[j]=x[j]*a;
9:          }
10:         for(k=0;k<16;k++){
11:                   c=c+y[k]*b;
12:         }
13:         for(l=0;l<16;l++){
14:                   d=d+z1[l];
15:         }
16:         for(m=0;m<16;m++){
17:                   z2[m]=c+x[m];
18:         }
19:         for(n=0;n<16;n++){
20:                   e=e+z2[n]/d;
21:         }
22: }
```

【図３】

```
                1: main()
                2: {
                3:          int i, a, b, x[16], y[16];
                4:          int x={2,7,3,6,9,10,1,5,6,6,13,17,9,4,7,8}
A1              5:          int y={1,4,7,2,5,8,3,6,9,15,16,1,7,7,0,0}
                6:          a=0;
A2              7:          for (i=0;i<16;i++){
                8:                    if(i<8){
A3              9:                              b=a+2;
                10:                   }
                11:                   else{
A4              12:                             b=a+1;
                13:                   }
B               14:                   function(a,b,x,y);
                15:         }
                16: }
```

【図５】

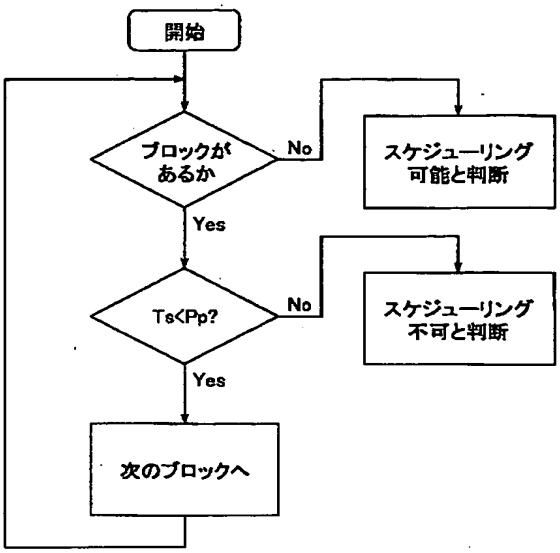


【図４】

```
1: function(a,b,x,y)
2: {
3:          int j, k, l, m, n, c, d, e, z1[16], z2[16];
4:          c=0;
5:          d=0;
6:          e=0;
7:          for(j=0;j<16;j++){
8:                      z1[j]=x[j]*a;
9:          }
10:         for(k=0;k<16;k++){
11:                     c=c+y[k]*b;
12:         }
13:         for(l=0;l<16;l++){
14:                     d=d+z1[l];
15:         }
16:         for(m=0;m<16;m++){
17:                     z2[m]=c+x[m];
18:         }
19:         for(n=0;n<16;n++){
20:                     e=e+z2[n]/d;
21:         }
22: }
```

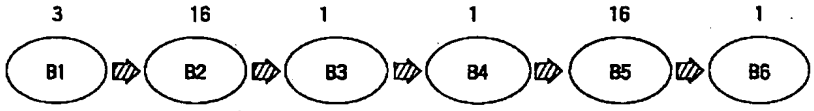
B1 / B2 / B3 / B4 / B5 / B6

【図１４】



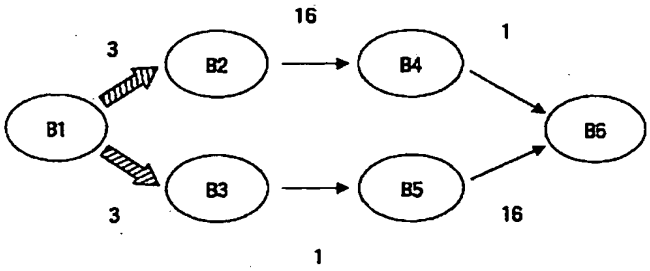
【図６】

function

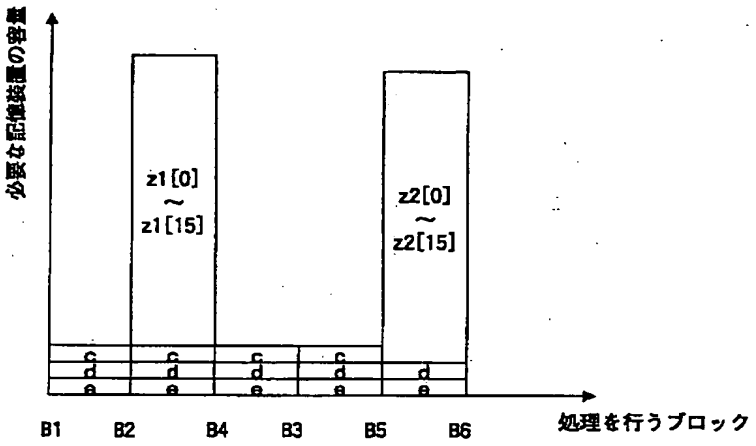


【図７】

function



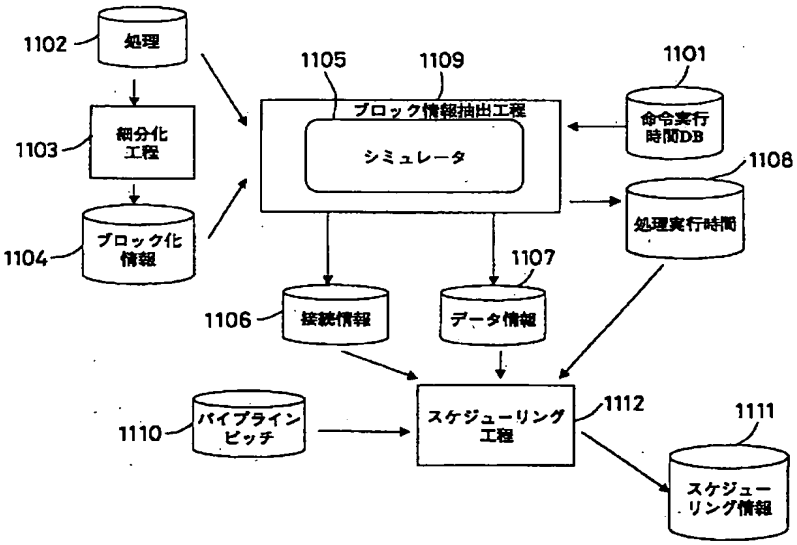
【図８】



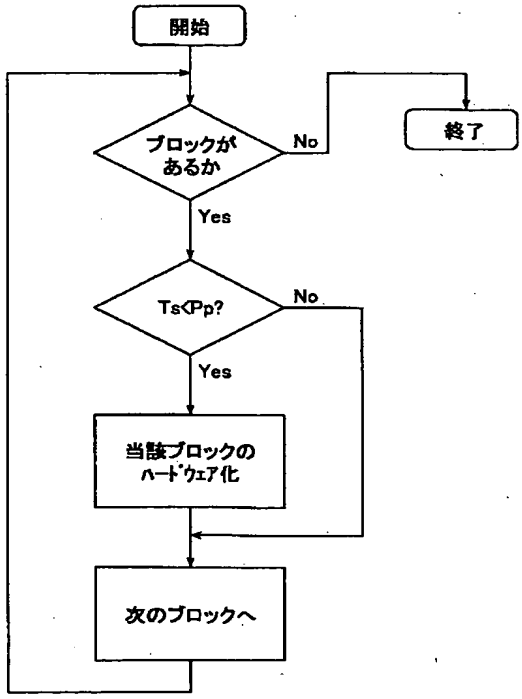
【図１０】

function

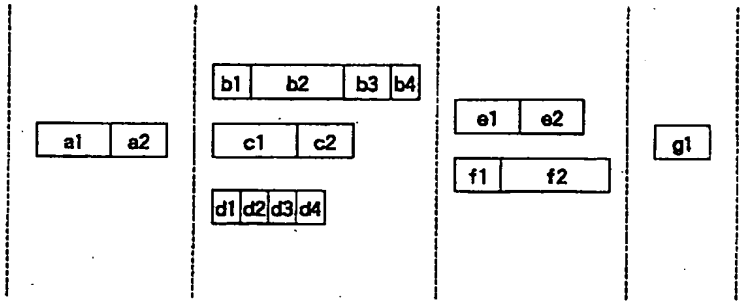
【図１１】



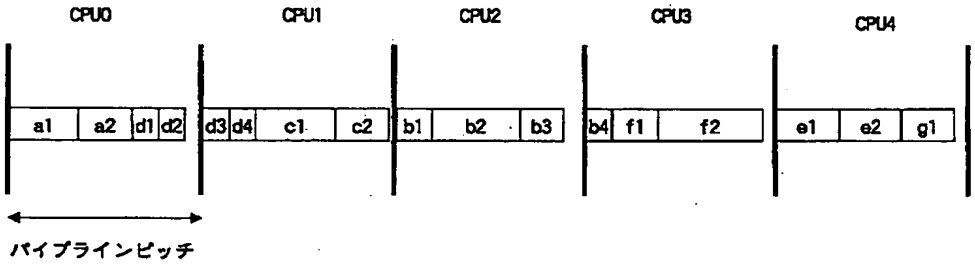
【図１６】



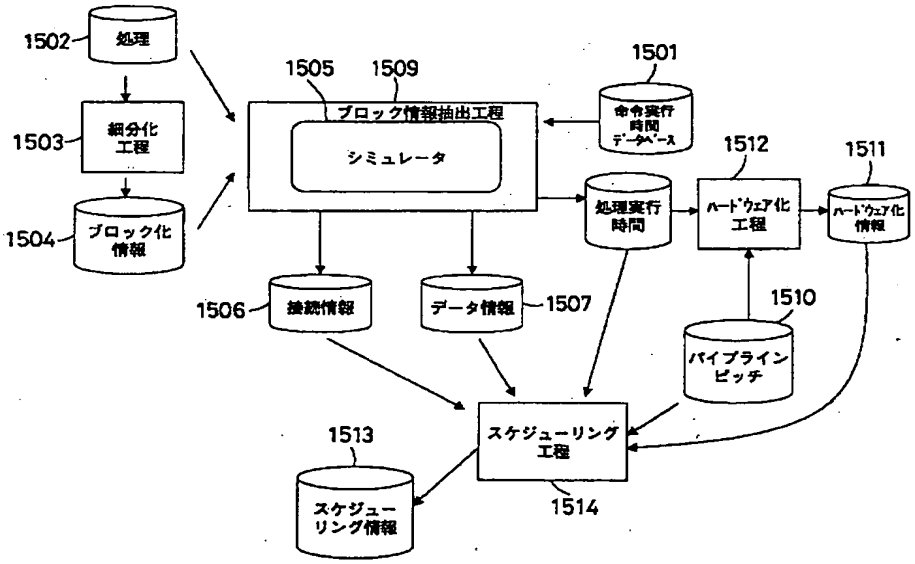
【図１２】


a1
a2
b1
b2
b3
b4
c1
c2
d1
d2
d3
d4
e1
e2
f1
f2
g1

【図１３】


CPU0
CPU1
CPU2
CPU3
CPU4
a1
a2
d1
d2
d3
d4
c1
c2
b1
b2
b3
b4
f1
f2
e1
e2
g1
パイプラインピッチ

【図１５】


1502
処理
1503
細分化工程
1504
ブロック化情報
1505
1509
ブロック情報抽出工程
シミュレータ
1501
命令実行時間データベース
1512
1511
処理実行時間
ハードウェア化工程
ハードウェア化情報
1506
接続情報
データ情報
1507
1510
パイプラインピッチ
スケジューリング工程
1513
スケジューリング情報
1514